

1. Prove the correctness of  $C_{IJ} = \sum_{K=1}^M A_{IK}B_{KJ}$

P.S. 本文的符号规定和课本相同，若为 column vector，则不加“ $\vec{\cdot}$ ”。若为 row vector，则加“ $\vec{\cdot}$ ”。

(pf)

Consider the four following cases:

**Case 1:** If  $B = [B_1 \ B_2]$ , where  $B_1$  is  $n \times 1$  and  $B_2$  is  $n \times (n-1)$ , then

$$AB = A(b_1, \dots, b_n, b_{n+1}, \dots, b_n)$$

$$= (Ab_1, \dots, Ab_n, Ab_{n+1}, \dots, Ab_n)$$

$$= (A(b_1, \dots, b_n), A(b_{n+1}, \dots, b_n))$$

$$= (AB_1, AB_2)$$

$$\text{Thus } A[B_1 \ B_2] = [AB_1 \ AB_2].$$

**Case 2:** If  $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ , where  $A_1$  is  $k \times n$  and  $A_2$  is  $(n-k) \times n$ , then

$$\begin{bmatrix} A_1 \\ A_2 \end{bmatrix} B = \begin{bmatrix} \vec{a}_1 \\ \vec{a}_2 \\ \vdots \\ \vec{a}_{k-1} \\ \vec{a}_n \end{bmatrix} B = \begin{bmatrix} \vec{a}_1 B \\ \vec{a}_2 B \\ \vdots \\ \vec{a}_{k-1} B \\ \vec{a}_n B \end{bmatrix}$$

$$= \begin{bmatrix} [\vec{a}_1] B \\ \vec{a}_2 B \\ \vdots \\ \vec{a}_{k-1} B \\ \vec{a}_n B \end{bmatrix} = \begin{bmatrix} A_1 B \\ A_2 B \end{bmatrix}$$

$$\text{Thus } \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} B = \begin{bmatrix} A_1 B \\ A_2 B \end{bmatrix}.$$

**Case 3** : Let  $A = [A_1 \ A_2]$  and  $B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$ , where  $\begin{cases} A_1 \text{ is } n \times s \text{ and } A_2 \text{ is } n \times (n-s), \text{ and} \\ B_1 \text{ is } s \times n \text{ and } B_2 \text{ is } (n-s) \times n. \end{cases}$

$$\text{If } C = AB, \text{ then } c_{ij} = \sum_{k=1}^n a_{ik} b_{kj} = \sum_{k=1}^s a_{ik} b_{kj} + \sum_{k=s+1}^n a_{ik} b_{kj},$$

↑  $a_{ik} b_{kj}$       ↑  $a_{ik} b_{kj}$       ↑  $a_{ik} b_{kj}$   
 $C_{B1B2}$        $A_1B_1 + A_2B_2$        $A_2B_2$

$$\text{Thus } [A_1 \ A_2] \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = AB = C = A_1B_1 + A_2B_2.$$

**Case 4** : Let  $A$  and  $B$  both partitioned as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}_{\frac{n}{s} \times \frac{k}{s}}, \quad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}_{\frac{s}{t} \times \frac{n}{t}}.$$

$$\text{Let } A_1 = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}, \quad A_2 = \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}, \quad B_1 = \begin{bmatrix} B_{11} & B_{12} \end{bmatrix}, \quad B_2 = \begin{bmatrix} B_{21} & B_{22} \end{bmatrix}.$$

It follows from **Case 3** that

$$AB = [A_1 \ A_2] \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} = A_1B_1 + A_2B_2$$

It follows from **Case 1** and **Case 2** that

$$A_1B_1 = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} B_1 = \begin{bmatrix} A_{11}B_{11} \\ A_{21}B_{11} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{bmatrix}$$

$$A_2B_2 = \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix} B_2 = \begin{bmatrix} A_{12}B_{21} \\ A_{22}B_{21} \end{bmatrix} = \begin{bmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{bmatrix}$$

Therefore,

$$\begin{aligned} AB &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \\ &= A_1B_1 + A_2B_2 = \begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{bmatrix} + \begin{bmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{bmatrix} \\ &= \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}. \end{aligned}$$

In general, if the blocks have proper dimensions, the block multiplication can be carried out the same way as ordinary matrix multiplication.

So if  $A_{nm}$  and  $B_{nm}$ , then

$$AB = \begin{bmatrix} C_{11} & \cdots & C_{1n} \\ \vdots & \ddots & \vdots \\ C_{m1} & \cdots & C_{mn} \end{bmatrix}, \text{ where } C_{IJ} = \sum_{K=1}^m A_{IK} B_{KJ}.$$

2. Show the numbers of additions and multiplications are the same.

看矩阵乘法的 code:

```
4 void matrixmul(float A[][SIZE], float B[][SIZE], float C[][SIZE]) {
5     for (auto i = 0; i < SIZE; i++) { // i: row index for C ← 每行，执行 SIZE 次
6         for (auto j = 0; j < SIZE; j++) { // j: col index for C ← 每列，执行 SIZE 次
7             C[i][j] = 0;
8             for (auto k = 0; k < SIZE; k++) C[i][j] += A[i][k] * B[k][j]; // k: iteration index
9         }
10    }
11
12    return;
13 }
```

因为是三重嵌套迴圈，所以最裡面的 statement "C[i][j] += A[i][k] \* B[k][j]" 需执行：

$$SIZE * SIZE * SIZE = (SIZE)^3 \xrightarrow{SIZE=n} n^3 \text{ 次。}$$

而因該此 statement "C[i][j] += A[i][k] \* B[k][j]" 包含了一个乘法和一个加法，所以  
 $\frac{1 \times n^3}{2} = \frac{n^3}{2}$  次。

再看 block matrix 版本的 code:

```
43 // compute C = A*B using blocking
44 void matrixmul_block2(float A[][SIZE], float B[][SIZE], float C[][SIZE], int p, int q) {
45     for (auto bi = 0; bi < SIZE; bi += p) { // bi: block row index for C ← 每行，执行  $\frac{SIZE}{p}$  次
46         for (auto bj = 0; bj < SIZE; bj += p) { // bj: block col index for C ← 每列，执行  $\frac{SIZE}{p}$  次
47
48             // clean C's value // 不太重要，只是把值全搞回0。
49             for (auto i = bi; i < bi + p; i++) { // i: row index for block
50                 for (auto j = bj; j < bj + p; j++) { // j: col index for block
51                     C[i][j] = 0.0;
52                 }
53
54             // compute the block submatrix C[bi][bj]
55             for (auto bk = 0; bk < SIZE; bk += q) { // bk: block index ← 每块，执行  $\frac{SIZE}{q}$  次
56                 for (auto i = bi; i < bi + p; i++) { // i: row index for block ← 每行，执行 p 次
57                     for (auto j = bj; j < bj + p; j++) { // j: col index for block ← 每列，执行 p 次
58                         for (auto k = bk; k < bk + q; k++) { // k: iteration index ← 每块，执行 q 次
59                             C[i][j] += A[i][k] * B[k][j];
60                         }
61                     }
62                 }
63             }
64         }
65     }
66 }
67 }
```

因為全都是巢狀迴圈，所以最裡面的 statement " $C[i][j] += A[i][k] * B[k][j];$ " 會執行：

$$\frac{\text{SIZE}}{P} * \frac{\text{SIZE}}{P} * \frac{\text{SIZE}}{q} * P * P * q = (\text{SIZE})^3 \frac{q}{P} = n^3 \text{ 次。}$$

而因為此 statement " $C[i][j] += A[i][k] * B[k][j];$ " 內包含了三個乘法和一個加法，所以

$$\begin{cases} \text{乘法} & 1 \times n^3 = n^3 \\ \text{加法} & 1 \times n^3 = n^3 \end{cases} \text{ 次。}$$

故兩種版本的  $(\text{乘法})$  的次數都完全一樣。

### 3. Implement the block version of matrix multiplication in C.

Skipped in this report.

### 4. Let $n=1024$ . Try different combinations of $p$ and $q$ , and measure their running times.

Time(seconds)	$p=4$	$p=8$	$p=16$	$p=32$	$p=64$	$p=128$	$p=256$			
$q=4$	3.989	3.453	3.249	3.200	3.182	3.158	3.162	遞減但最後又高一點		
$q=8$	3.651	3.335	3.272	3.223	3.208	3.200	3.205	遞減但最後又高一點		
$q=16$	3.628	3.404	3.319	3.280	3.258	3.257	3.261	遞減但最後又高一點		
$q=32$	3.608	3.376	3.335	3.283	3.268	3.264	3.267	遞減但最後又高一點		
$q=64$	3.613	3.394	3.333	3.304	3.296	3.299	3.300	遞減但最後兩個又高一點		
$q=128$	3.677	3.479	3.393	3.384	3.364	3.354	3.346	遞減		
$q=256$	3.732	3.528	3.452	3.433	3.376	3.370	3.382	遞減但最後又高一點		
	遞減到 $q=32$	遞增到 $q=32$	遞增	遞增	遞增	遞增	遞增			
再遞增	再遞增	有突低								
	有突高									

P.S. 上表格中的每個元素，都是算 10 次後取到的平均值。

5. Google what is cache memory in computer architecture, and look up the cache size of your computer. Explain the reason of the performance differences for various p and q.

因為array是我們常用的資料結構，在這當中，又以連續的index的存取最為常用。

所以，CPU為了加快存取的速度(減少匯流排和RAM之間的往返)，在自己內部有一塊空間可以儲存資料。這個空間叫做cache。

我的CPU的cache 是 L1=256KB、L2=1MB、L3=6MB。

因為有三個array，每個array都是 $1024 \times 1024 (=1K \times 1K = 1M)$ 個int(每個int占用4 bytes)，所以如果三個矩陣都要儲存進去的話，總共花費 $1M \times 4B \times 3 = 12MB$ ，顯然所有cache加起來都不夠。

而剛剛有提到，我們常常用連續的index，所以CPU會自動抓取在RAM中連續的index的資料來抓到cache裡，稱為cache line。一個cache line依照CPU廠商的設計，可以是32or64or128 bytes不等。

這個做法對於「巨大的」矩陣的乘法就很有問題了。因為矩陣乘法是「一橫列(row) 和 一直行(column)」內積，row的部分可以cache line沒問題(儘管因為cache line很小，所以會被切成很多次的cache line，但都是有效利用)。但column的部分，因為array是2D的緣故，所以cache line雖然存在，但完全沒有發揮到應有的功能，導致變得很慢。

有一種加快速度的方式，就是把AB矩陣中的B進行transpose，這樣，「一橫列(row) 和 一直行(column)」內積，就會變成「一橫列(row) 和 另一橫列(row)」內積(或者一直行 和 另一直行)內積。如此，A和B-t(transposed B)就可能可以完美搭配兩個有效利用的cache line。為啥說可能可以，是因為cache line的設計到底是取row還是取column 可能會由個別CPU來影響。

但transpose的方法，還是有：「一個row過長所以要分很多次cache line」的問題。

有沒有辦法可以直接把cache line很fit給「小row」或「小column」呢？這就是本次作業block matrix multiplication的範疇。

Block Matrix Multiplication就是把「C=AB」矩陣的解，不再以傳統數學的方式，算一個元素就一次性地把一個row和一個column內積。而是把row和column都有效分段，把每個分段的內積都算好之後，再把所有分段的內積都加起來，就是該元素的最終解答。C的每個元素都走過這個流程，就可以得到完整的解答。

那，怎麼樣的分段方式(p, q的組合)會是最有效率的呢？

(因為是經由實驗統計歸納，所以可能有個別組合並無依照下列的規則，可能是因為採樣數不夠多導致的誤差。)

一、在p相同時，q越大，所需的時間越長。

二、在q相同時，p越大，所需的時間越短

綜合上述兩點來看，若p越大且q越小，在計算上是最快的。表格右上方是整個表格中，少數會出現「3.2xx，且接近3.200」，或者「3.1xx」的地方，即可證明此歸納。

實驗環境是Windows10家用版1909，Visual Studio 2017 C++17 x86的Release模式。